Incremental Concept Formation made More Efficient by the Use of Associative Concepts

Sébastien Ferré

▶ **To cite this version:**

Sébastien Ferré. Incremental Concept Formation made More Efficient by the Use of Associative Concepts. [Research Report] RR-4569, INRIA. 2002. inria-00072019

HAL Id: inria-00072019
https://hal.inria.fr/inria-00072019

Submitted on 23 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Incremental Concept Formation made More Efficient by the Use of Associative Concepts

Sébastien Ferré

N˚4569

October 2002

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4569--FR+ENG

INRIA
RENNES

Incremental Concept Formation made More Efficient by the Use of Associative Concepts

Sébastien Ferré*

Thème 2 — Génie logiciel
et calcul symbolique
Projet Lande

Rapport de recherche n ° 4569 — October 2002 — 13 pages

Abstract: Formal Concept Analysis (FCA) is interested in the formation of concept lattices from binary relations between objects and attributes, a.k.a. contexts. Many algorithms have been proposed to generate the set of all concepts, and also the edges of the lattice between these concepts. We develop the principle and the code of a new algorithm combining two existing ones, Godin's and Bordat's algorithms. Then, we show by both a theoretical and practical study that it is the most efficient algorithm for sparse contexts, which are usually found in real applications.

Key-words: context, concept lattice, Galois lattice, incremental algorithm, complexity

(Résumé : tsvp)

* This author is supported by a scholarship from CNRS and Région Bretagne.

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

La formation incrémentale de concepts rendue plus efficace par l'emploi de concepts associatifs

Résumé : L'Analyse de Concepts Formelle (ACF) s'intéresse à la formation de treillis de concepts à partir de relations binaire entre des objets et des attributs, appelées contextes. De nombreux algorithmes ont été proposés pour générer l'ensemble des concepts, ainsi que les arcs du treillis entre ces concepts. Nous développons le principe et le code d'un nouvel algorithme combinant deux algorithmes existants, ceux de Godin et de Bordat. Puis, nous montrons par une étude à la fois théorique et pratique, que c'est l'algorithme le plus efficace pour les contextes creux, que l'on trouve habituellement dans les applications réelles.

Mots-clé : contexte, treillis de concepts, treillis de Galois, algorithme incrémental, complexité

1 Introduction

Formal Concept Analysis (FCA, [Wil82, GW99]) is interested in the formation of *concepts* from a binary relation I between a set of *objects* $\mathcal{O}$ and a set of *attributes* $\mathcal{A}$: the triple $K = (\mathcal{O}, \mathcal{A}, I)$ is called a *context*. The concepts of such a context are the elements of the Galois lattice [DP90], corresponding to the Galois connection (σ, τ) defined by

$$\begin{array}{lcll} \sigma_K(O) & = & \bigcap_{o \in O} I(o), & \text{for all } O \subseteq \mathcal{O}, \text{ and} \\ \tau_K(A) & = & \{o \in \mathcal{O} \mid I(o) \supseteq A\}, & \text{for all } A \subseteq \mathcal{A}. \end{array}$$

Therefore, concepts are pairs (O, A), where O is a set of objects (the *extent*) and A is a set of attributes (the *intent*), such that $\sigma_K(O) = A$ and $\tau_K(A) = O$. The ordering on concepts in the Galois lattice is defined by

$$(O_1, A_1) \leq (O_2, A_2) \iff O_1 \subseteq O_2 \quad (\iff A_1 \supseteq A_2).$$

In FCA, the Galois lattice of some context K is usually called the *concept lattice* and is denoted by L_K.

This concept lattice has been found useful in many domains, such as software engineering [KS94, Sne98, ST00], information retrieval [GMA93, Lin95, CS00, FR01], and knowledge extraction [HSWW00]. Several algorithms have been designed to compute the set of concepts or even the Hasse diagram of the concept lattice. A comparative study, both theoretical and practical, can be found in [KO01].

Among these algorithms, some are *incremental* [GMA95, VM01], which means they can update the concept lattice when a new object is added to the context, without re-computing the whole lattice. On the contrary, *batch* algorithms have to know the whole context before computing the concept lattice; if the context changes, the concept lattice must be re-computed entirely. Incremental algorithms are especially interesting in information systems where the set of objects is growing all the time.

In a previous work [FR02], we designed an incremental learning process that helps users in describing/classifying new objects according to the description/classification of existing objects. We also established a connection between this incremental learning process and the incremental concept formation, which let us think that a more efficient algorithm could be designed for incremental concept formation.

Section 2 explains the above connection and draws the principle of our algorithm. Section 3 details this algorithm, along with data structures. Section 4 evaluates the theoretical complexity of our algorithm and of two other well-known efficient algorithms (Godin's and Bordat's) to compare them under similar conditions. It proves our algorithm is the most efficient one when the number of attributes per object is bounded (which is often observed in practice). It also discusses complexity for different hypotheses on contexts. Section 5 performs practical experiments with these three algorithms in order to complete the theoretical comparison. Finally, Section 6 summaries our results and draws a few perspectives.

2 Principle for a New Incremental Algorithm

In this section, we compare the incremental concept formation [GM94, GMA95, VM01] with the incremental learning process [FR02]. Both methods are based on the search for specific concepts: old, modified, generator, and new concepts in the first case (see Section 2.1); associative concepts in the second case (see Section 2.2). Surprisingly, we found a close relationship between associative concepts and both modified and new concepts (see Section 2.3).

2.1 Incremental Concept Formation

Incremental Concept Formation (ICF) starts with a context $K = (\mathcal{O}, \mathcal{A}, I)$, whose concept lattice L_K had already been computed. The problem is to compute the lattice L_{K^*}, where K^* is the result of adding a new object o^* described by the set of attributes $D(o^*) = \sigma_K(\{o^*\})$:

$$K^* = (\mathcal{O} \uplus \{o^*\}, \mathcal{A} \cup D(o^*), I \uplus \{(o^*, a) \mid a \in D(o^*)\}).$$

The principle is that every concept intent of L_K is still a concept intent of L_{K^*}; and every new concept intent is the intersection of some old concept intent and $D(o^*)$. Moreover, the new object o^* may be added to the extent of some old concepts. Therefore, concepts of L_K can be classified in 3 categories:

generator concepts : the intersection of their intent with D results in a new intent, and so, in a **new concept**;

modified concepts : their intent is included in $D(o^*)$, and so, o^* must be added to their extent;

old concepts : all other concepts, which are kept unchanged.

We now give a formal definition for generator and new, modified, and old concepts.

Definition 1 (generator and new concepts) *The* new *concepts are those whose intent is the intersection of the intent of an existing concept (a* generator *concept) and of the description $D(o^*)$ of the new object, and does not already exist: $N(o^*) = \{(ext, int \cap D(o^*)) \mid \exists (ext, int) \in L_K : (int \cap D(o^*)) \notin int(L_K)\}$.*

Definition 2 (modified concepts) *The* modified *concepts are those whose intent is included in the description $D(o^*)$: $M(o^*) = \{(ext, int) \in L_K \mid D(o^*) \supseteq int\}$.*

Definition 3 (old concepts) *The* old *concepts are those that are neither generator, nor modified concepts.*

The update of a concept lattice L_K due to inserting a new object o^* consists in inserting all new concepts, and by adding o^* to the extension of all modified and new concepts. Therefore, the main task of incremental concept formation is to find all modified and new concepts.

2.2 Incremental Learning Process

Incremental Learning Process (ILP) starts with the same data as ICF, but its aim is quite different. The aim of ILP is to induce a set of attributes $Ind(o^*)$ that seem to fit to the new object, according to the description of existing objects. Then, these induced attributes can be suggested to the user, or automatically added to the description $D(o^*)$ of the new object. The definition of $Ind(o^*)$ is based on hypotheses [Kuz99, GK00] according to the JSM method [Fin83]. There also exists a characterisation based on the notion of *associative* concepts.

Definition 4 (sub-context) *Let $K = (\mathcal{O}, \mathcal{A}, I)$ be a context, and $A \subseteq \mathcal{A}$ be a subset of attributes. We define the* sub-context *of K restricted to the subset of attributes A, the context*

$$K(A) = (\mathcal{O}, A, I \cap (\mathcal{O} \times A)).$$

Definition 5 (associative concept) *A non-empty concept of the sub-context $K(\mathcal{A} \cap D(o^*))$ is called an* associative concept *of o^* in K. The set of all such associative concepts is denoted by $AC(o^*)$.*

$AC(o^*)$ organizes the context K in a concept lattice (where the empty concept is missing) that is less finely detailed than L_K. However, this coarser concept lattice is relevant to the attributes of o^*. Conversely, the finer details in L_K cannot be expressed with the attributes of o^*.

Then, an *induced feature* can be defined as an attribute that contextually subsumes the intent of some associative concepts.

Definition 6 (induced property) *We say an attribute x is an induced property iff there exists an associative concept $c \in AC(o^*)$, such that $ext(c) \subseteq \tau_K(x)$. $Ind(o^*)$ denotes the set of all induced properties of o^* in K.*

Theorem 1 $Ind(o^*) = \bigcup_{(ext,int) \in AC(o^*)} \sigma_K(ext)$.

Intuitively, an associative concept c of a new object o^* is an already existing concept (for previous objects in K) that has some similarity with the description of o^*. When $x \in Ind(o^*)$ is induced from an associative concept c, $ext(c)$ is the *support* of the induction, and $int(c)$ is the *explanation*. A given associative concept can induce several attributes; and a given attribute can be induced by several associative concepts, and so, have several explanations. Induced attributes that do not belong to the description $D(o^*)$ are called *expected features*, which are the features suggested to users.

2.3 Connection between Incremental Concept Formation and Incremental Learning Process

The connection between ICF and ILP is that associative concepts match exactly new and modified concepts, with the exception of the empty concept. These connection is stated and proved in the two following theorems.

Lemma 1 (sub-context Galois connection) *Let $K = (\mathcal{O}, \mathcal{A}, I)$ be a context, and $D \subseteq \mathcal{A}$,*

- $\sigma_{K(D)}(O) = \sigma_K(O) \cap D$, *for all* $O \subseteq \mathcal{O}$;
- $\tau_{K(D)}(A) = \tau_K(A)$, *for all* $A \subseteq \mathcal{A}$.

Proof:
• $\sigma_{K(D)}(O) = \{a \in D \mid \forall o \in O : (o,x) \in I \cap (\mathcal{O} \times D)\}$
$= \{a \in \mathcal{A} \mid a \in D \text{ and } \forall o \in O : (o,a) \in I\}$
$= \sigma_K(O) \cap D$.
• $\tau_{K(D)}(A) = \{o \in \mathcal{O} \mid \forall a \in A : (o,a) \in I \cap (\mathcal{O} \times D)\}$
$= \{o \in \mathcal{O} \mid \forall a \in A : (o,a) \in I\}$ because $A \subseteq D$
$= \tau_K(A)$. ■

Theorem 2 proves that non-empty modified concepts are the associative concepts such that their intent is equal to the closure of the extent by σ_K, i.e. exists in L_K.

Theorem 2 *The following statements are equivalent:*

1. $c \in M(o^*)$ *and* $ext(c) \neq \emptyset$, *and*
2. $c \in AC(o^*)$ *and* $\sigma_K(ext(c)) = int(c)$.

Proof:
• $c \in M(o^*)$ and $ext(c) \neq \emptyset$
$\Longrightarrow c \in L_K$ and $int(c) \subseteq D(o^*)$ Def. 2
We have $\sigma_{K(D(o^*))}(ext(c)) = \sigma_K(ext(c)) \cap D(o^*)$ Lemma 1
$= int(c) \cap D(o^*) = int(c)$.
Hence $c \in L_{K(D(o^*))}$ and $\sigma_K(ext(c)) = int(c)$
$\Longrightarrow c \in AC(o^*)$ and $\sigma_K(ext(c)) = int(c)$.
• $c \in AC(o^*)$ and $\sigma_K(ext(c)) = int(c)$
$\Longrightarrow c \in L_{K(D(o^*))}$ and $ext(c) \neq \emptyset$ Def. 5
Since we have also $\tau_K(int(c)) = \tau_{K(D(o^*))}(int(c)) = ext(c)$,
we have $c \in L_K$ and $int(c) \subseteq D(o^*)$
$\Longrightarrow c \in M(o^*)$ and $ext(c) \neq \emptyset$. ■

Theorem 3 proves that non-empty new concepts are the associative concepts such that the intent is not equal to the closure of the extent by σ_K, i.e. does not exist in L_K.

Theorem 3 *The following statements are equivalent:*

1. $c \in N(o^*)$ *and* $ext(c) \neq \emptyset$, *and*
2. $c \in AC(o^*)$ *and* $\sigma_K(ext(c)) \neq int(c)$.

Proof:
• $(\tau_K(int), int) \in N(o^*)$ and $\tau_K(int) \neq \emptyset$
$\Longrightarrow \exists c' \in L_K : int = int(c') \cap D(o^*)$ and $\sigma_K(\tau_K(int)) \neq int$ (that is, int is not an intent of L_K).
We now show that int is an intent of $L_{K(D(o^*))}$:
$\sigma_{K(D(o^*))}(\tau_{K(D(o^*))}(int))$
$= \sigma_{K(D(o^*))}(\tau_{K(D(o^*))}(\sigma_K(ext(c')) \cap D(o^*)))$
$= \sigma_{K(D(o^*))}(\tau_{K(D(o^*))}(\sigma_{K(D(o^*))}(ext(c'))))$
$= \sigma_{K(D(o^*))}(ext(c')) = \sigma_K(ext(c')) \cap D(o^*)$
$= int(c') \cap D(o^*) = int$.
Hence, if we define $c = (\tau_K(int), int)$ we have
$c \in L_{K(D(o^*))}$ and $ext(c) \neq \emptyset$ and $\sigma_K(ext(c)) \neq int(c)$
$\Longrightarrow c \in AC(o^*)$ and $\sigma_K(ext(c)) \neq int(c)$.
• $c \in AC(o^*)$ and $\sigma_K(ext(c)) \neq int(c)$
$\Longrightarrow c \in L_{K(D(o^*))}$ and $ext(c) \neq \emptyset$.
We show that $ext(c)$ is an extent of L_K: we have
$\tau_{K(D(o^*))}(\sigma_{K(D(o^*))}(ext(c))) = ext(c)$ ($ext(c)$ is an extent of $L(K(D(o^*)))$)
$\Longrightarrow \tau_K(\sigma_K(ext(c)) \cap D(o^*)) = ext(c)$

$\Longrightarrow \tau_K(\sigma_K(ext(c))) = \tau_K(\sigma_K(ext(c) \cap D(o^*))) = ext(c)$.
Hence $(ext(c), \sigma_K(ext(c))) \in L_K$.

Furthermore, $int(c) = \sigma_{K(D(o^*))}(ext(c)) = \sigma_K(ext(c)) \cap D(o^*)$,
hence $\exists c' = (ext(c), \sigma_K(ext(c))) \in L_K : int(c) = int(c') \cap D(o^*)$
and $int(c) \notin int(L_K)$ (because $\sigma_K(ext(c)) \neq int(c)$)
$\Longrightarrow c \in N(o^*)$ and $ext(c) \neq \emptyset$. ■

To summarize, the non-empty modified and new concepts are exactly the associative concepts. There is an empty modified concept when $D(o^*) \supseteq \mathcal{A}$ and it is $(\emptyset, \mathcal{A})$. There is an empty new concept when $\tau_K(D(o^*)) = \emptyset$ and it is $(\emptyset, D(o^*))$. So, it is sufficient to traverse the concept lattice of $K(D(o^*))$ instead of the whole context K, because of the definition of associative concepts (Def. 5). By the way, we avoid the case where two generator concepts are found but only one new concept is generated. This suggests that the incremental concept formation proposed by Godin et al. could be optimized by the use of associative concepts.

2.4 Informal Description of a New Algorithm

We now informally describe our algorithm. The idea is to generate all associative concepts $AC(o^*)$, and to test for each of them if it is modified or new. If it is modified, i.e., if it already exists in L_K, then the new object o^* must be added to its extent. If it is new, i.e., if it does not exist in L_K, the new object o^* must be added to its extent, and it must be inserted in the concept lattice. Then, if no concept has $D(o^*)$ as intent, a concept $(\{o^*\}, D(o^*))$ must also be inserted. Finally, when all objects have been inserted, and if no concept has $\mathcal{A}$ as intent, a concept $(\emptyset, \mathcal{A})$ must also be inserted (bottom concept).

Compared to Godin's algorithm, we replace the traversal of the whole concept lattice L_K by the generation of the coarser concept lattice $L_{K(D(o^*))}$. The advantage of our approach is that the set of considered attributes is restricted to $D(o^*)$, the attributes of the new object, which has often in practice a bounded size that does not depend on the number of objects and attributes. The drawback is that we have to generate associative concepts, whereas Godin reuses directly the concept lattice already computed.

So, to get a chance to better other algorithms, we should find the most appropriate algorithm to generate associative concepts $AC(o^*)$, that is, the concept lattice of $K(D(o^*))$. Because we noted before that the size of $D(o^*)$ is often bounded in practice, we should prefer the best algorithm when the number of attributes is considered as a constant. From a survey on concept lattice algorithms [KO01], the good choice seems to be Bordat's algorithm [Bor86] because it is only linear in the number of objects, whereas other algorithms are quadratic. This is a batch top-down algorithm. It starts from the top concept, i.e., whose extent contains all objects, and successively builds sub-concepts, while checking if a built concept had already been found.

3 A New Incremental Algorithm for Concept Formation

In this section, we details data structures, procedures and functions of our incremental algorithm for the computation of the concept lattice. The global data structures represent the context and its concept lattice. The main procedure, `add-object`, updates these structures with a new object `o`, described by `D`. For efficiency reasons, the bottom concept is not considered in this procedure, but it can be inserted by procedure `add-bottom`, after all objects have been inserted. The lines beginning with a ◇ concern the building of the Hasse diagram of the concept lattice. They can be dropped if one is only interested in the concept set.

Considering that `obj` is the type of objects, and `attr` is the type of attributes, we respectively represent extents and intents by `obj set` and `attr set`. For building the concept set, we use an array `col` to represent columns of the context (i.e., extents of attributes), a set of attributes `top` to represent the intent of the top concept, and a hashtable `C` of concepts indexed by their intents. For building the Hasse diagram, we also use an array `row` to represent rows of the context (i.e., descriptions of objects), and another hashtable `Sub` that associates to each intent of concept the intents of their sub-concepts.

◇ **data** `row` : **array**`[1..|`$\mathcal{O}$`|]` **of** `attr set` = `[`$\emptyset$`;..;`$\emptyset$`]`

data `col` : **array**`[1..|`$\mathcal{A}$`|]` **of** `obj set` = `[`$\emptyset$`;..;`$\emptyset$`]`

data `top` : `attr set` = $\emptyset$

data `C` : **hashtable**`[attr set]` **of** `obj set` = $\emptyset$

◇ **data** Sub : **hashtable**[attr **set**] **of** attr **set** **set** = $\emptyset$

col[a] denotes the context column of attribute a. C[Int]? is true if the pair (Ext,Int) belongs to the hashtable C; and C[Int] denotes the extent associated to intent Int (when C[Int]? is true). In the following procedures, $A \leftarrow B$ denotes the assignment of B to A.

We now give the procedure add-object, which adds object o with description D. It first calls procedure iter-assoc to perform the generation of associative concepts. If D is not yet defined in C, the concept ({o},D) is inserted (see Section 2.4). Finally, a row is added and columns are updated.

```
procedure add-object(o : obj; D : attr set)
    iter-assoc(o,D);
    if not C[D]? then
       Ext ← Ext ∪ {({o},D)};
       ◇ Sub ← Sub ∪ {(D,∅)};
    ◇ row[o] ← D;
    for all a ∈ D do
       col[a] ← col[a] ∪ {o}.
```

Procedure iter-assoc initiates the generation of associative concepts of the new object o; procedure iter-assoc-2 is recursive and completes this generation. This generation is inspired from the algorithm of Bordat. In iter-assoc-2, X is the set of attributes that must be considered to find sub-concepts of the current associative concept (Ext,Int); Incr is a selection of those attributes of X that restrict the current extent Ext without making it empty; and S is the set of sub-concepts built with the maximal elements of Incr by function sub-nodes.

```
procedure iter-assoc(o : obj; D : attr set)
    if C[top]? then /* every object except the first one */
       Int ← D ∩ top;
       iter-assoc-2(o, D, true, C[top], Int, D \ Int);
    else /* first object */
       iter-assoc-2(o, D, true, ∅, D, ∅).

procedure iter-assoc-2(o : obj; D : attr set; is-top : bool; Ext : obj set; Int : attr
    set; X : attr set)
    if C[Int]? then /* (Ext,Int) is a modified concept */
       C[Int] ← Ext ∪ {o};
    else /* (Ext,Int) is a new concept */
       C ← C ∪ {(Ext ∪ {o},Int)};
       ◇ if Ext = ∅ then
       ◇    Sub ← Sub ∪ {(Int, ∅)};
       ◇ else
       ◇    Sub ← Sub ∪ {(Int, ⋂_{o'∈Ext} row[o'])};
       if is-top then top ← Int;
    Incr ← {(Ext ∩ col[x], x) | x ∈ X, Ext ∩ col[x] ≠ ∅};
    ◇ if Incr = ∅ then
    ◇    if Int ≠ D then Sub[Int] ← insert-sub(D, Sub[Int]);
    ◇ else
       S ← sub-nodes(Incr, Int);
       for all (Ext',Int') ∈ S do
          if not C[Int]? or o ∉ C[Int] then
             iter-assoc-2(o, D, false, Ext', Int', {x | (-,x) ∈ Incr, x ∉ Int'});
          ◇ Sub[Int] ← insert-sub(Int', Sub[Int]).

◇ function insert-sub(Int : attr set; Sub : attr set set) : attr set set
    ◇ return {Int' ∈ Sub | Int' ⊈ Int} ∪ {Int}

function sub-nodes(Incr : (obj set, attr) set; Int : attr set) : (obj set, attr set) set
    A ← Incr; S ← ∅;
    for all (Ext,a) ∈ Incr do
```

```
      A ← A \ {(Ext,a)}; max ← true; Equal ← {a};
      for all (Ext',a') ∈ A do
         if Ext ⊊ Ext' then max ← false; Stop
         if Ext = Ext' then
            A ← A \ {(Ext',a')}; Equal ← Equal ∪ {a'};
         else /* Ext ⊋ Ext' */
            A ← A \ {(Ext',a')};
      if max = true then S ← S ∪ {(Ext,Int ∪ Equal)};
   return S.
```

The bottom concept has the set of all attributes as intent. If this intent is not defined in `C`, procedure `add-bottom` adds the concept $(\emptyset, \mathcal{A})$, and completes the linking of the Hasse diagram if necessary.

procedure add-bottom($\mathcal{A}$: attr **set**)
  if not C[$\mathcal{A}$]? **then**
    C ← C ∪ {($\emptyset$,$\mathcal{A}$)};
    ◇ Sub ← Sub ∪ {($\mathcal{A}$,$\emptyset$)};
    ◇ **for all** (Int,S) ∈ Sub **do**
    ◇ **if** S = $\emptyset$ **then** Sub[Int] ← {$\mathcal{A}$}.

This algorithm was implemented in Objective Caml, a programming language that combines functionnal, imperative and object features. Attribute sets and object sets are represented by ordered lists, which makes all set operations linear with the size of sets. The hash function on intents is considered as linear in the size of intents, which determines the cost of operations on hashtables.

4 Discussion on Theoretical Complexity depending on Contexts

In this section, we discuss the theoretical complexity of our algorithm, and compare it with Bordat's and Godin's. To express theoretical complexities, we retain 3 parameters that determine the size of contexts:

n: the number of objects;

m: the number of attributes;

k: the maximal number of attributes per object.

This last parameter is not usually used in theoretical complexities, but is often invoked to say that the size of the concept lattice is polynomial with the number of objects n, when k is bounded. Indeed in this case, the number of concepts, which we denote by l, is in $\mathcal{O}(n2^k) = \mathcal{O}(n)$: each object has at most 2^k concepts above it (maximal number of subsets of its description). Even if in general the number of concepts may be exponential in the number of objects or the number of attributes, we think it is preferrable to express theoretical complexities with parameter k. This makes them more precise.

4.1 Bordat's and Godin's algorithms

So, to get fair comparisons with other algorithms, we must first reconsider their complexity with regard to parameter k. For instance, Bordat's algorithm is known to have a complexity in $\mathcal{O}(nm^2l)$. Should we replace the occurence of m (number of attributes) by an occurence of k (number of attributes per object)? Here, the answer is no because m corresponds to the traversing of all attributes. About Godin's algorithm, who has a complexity in $\mathcal{O}(n(n+m)l)$, should we replace m by k? Here, the answer is yes because this occurence of m corresponds to set operations on intents, whose size is bounded by k. So, we now consider that Godin's has a complexity in $\mathcal{O}(n(n+k)l)$.

4.2 Our algorithm

In our algorithm, either `iter-assoc` or `iter-assoc-2` is called on all associative concepts generated through the insertion of all objects. By a close examination of these two procedures, we observe that `iter-assoc-2` has a greater complexity than `iter-assoc`. The complexity of the former is determined by the complexity of functions `sub-nodes` and `insert-sub` (called for (at worst) each attribute of the new object):

`sub-nodes` pairwise compares the extent of the attributes of the new object: its (worst-case) complexity is therefore in $\mathcal{O}(k^2n)$;

`insert-sub` compares some intent with the intents of a set of sub-concepts. As the number of sub-concepts is bounded by m, its complexity is in $\mathcal{O}(km)$.

Therefore, the complexity of `iter-assoc-2` is in $\mathcal{O}(k^2(n+m))$. So, the whole theoretical complexity of our algorithm is in $\mathcal{O}(k^2(n+m)a)$, where a denotes the whole number of associative concepts through the insertion of all objects.

To compare this complexity with other algorithms, we must relate in some way parameter a to parameter l. Firstly, the number of associative concepts $AC(o^*)$ for every new object o^* is bounded by the number of subsets of its description $D(o^*)$, that is 2^k: so, $a \leq 2^k n$. Secondly, if we state the reasonable assumption that every objects have pairwise different description, the inequality $n \leq l$ is satisfied because every object description is a concept intent. In the other case, duplicate objects can easily be identified (their description is already an intent), and inserted (add the new object to every extent containing the former object). In conclusion, we can safely replace a by the term $2^k l$.

Thirdly, a is also equal to the number of object insertions in concept extents (see `iter-assoc-2`), and so, is equal to the sum of the size of all concept extents, which is in $\mathcal{O}(nl)$. So, we can also safely replace a by the term nl. Finally, we can give to our algorithm a complexity in $\mathcal{O}(k^2 min(2^k, n)(n+m)l)$.

4.3 Comparisons

To compare Bordat's, Godin's, and our algorithms, we consider 4 cases for complexity:

- the general case, where complexity is expressed with k, n, and m;
- the case where k may be as big as m (i.e., $k = m$);
- the case where k is a constant independent of n and m;
- the case where k is a constant, and m grows proportionally with n (i.e., $m = \alpha n$).

In practice, we have observed that, while the number of objects is growing, the number of attributes per object remains bounded, which justifies to consider k as a constant in the 3rd and 4th cases. Of course, the value of k strongly depends on the considered application. Moreover, we have also observed that often, each object brings with it a few new attributes, such that the number of attributes grows as much as the number of objects: this justifies to consider m as equal to n in the 4th case.

The 4 cases of complexities for the 3 algorithms are summarized in the following table:

complexity	Bordat	Godin	Ferré
general case	nm^2l	$n(n+k)l$	$k^2 min(2^k, n)(n+m)l$
$k \in \mathcal{O}(m)$	nm^2l	$n(n+m)l$	$n(n+m)m^2l$
$k \in \mathcal{O}(1)$	nm^2l	n^2l	$(n+m)l$
$k \in \mathcal{O}(1), m \in \mathcal{O}(n)$	n^3l	n^2l	nl

It appears that, while our algorithm is worse than other ones in non-sparse contexts (i.e., $k \in \mathcal{O}(m)$), it is strictly better in sparse contexts. It is better than Bordat's because we exploit the bounded size of intents, and we carfully avoid to traverse the set of all attributes. It is better than Godin's because we replaced the n traversal of the lattice by the traversal of associative concepts, whose number is bounded by $2^k l$, which is in $O(l)$ when k is a constant.

5 Practical Complexity

To complete the theoretical studies of the complexity of our algorithm compared to Bordat's and Godin's, we present in this section a practical comparison of these algorithms. We can consider the computation of either the set of concepts or the Hasse diagram of concepts, i.e., the set of concepts plus links berween concepts [KO01]. For Bordat's algorithm, there is no substantial difference since the computation of the set of concepts implies the computation of all sub-concepts of each concept. For Godin's most efficient algorithm, the computation of links is not sufficiently detailed to be coded, unfortunately. So, we compare in this article the three algorithms

for computing the set of concepts. However, we also present the pratical complexity of our algorithm when computing the Hasse diagram. Note that this is to the advantage of Bordat's and Godin's, especially for the latter, whose general complexity is then in $\mathcal{O}(knl)$. Note also that if a better version of Bordat's algorithm were used, this would also benefit to us because our algorithm uses it nearly directly.

5.1 General contexts

The aim of this sub-section is to study how pratical complexities depend on each of the three parameters n, m, and k. For each valuation of this triple, a formal context is generated randomly with n objects whose descriptions are composed of k out of m attributes. In each of the three following figures, the computation time is figured in function of the number of objects n (Figure 1), the number of attributes m (Figure 2), and the number of attributes per object k (Figure 3, logarithmic scale for time).

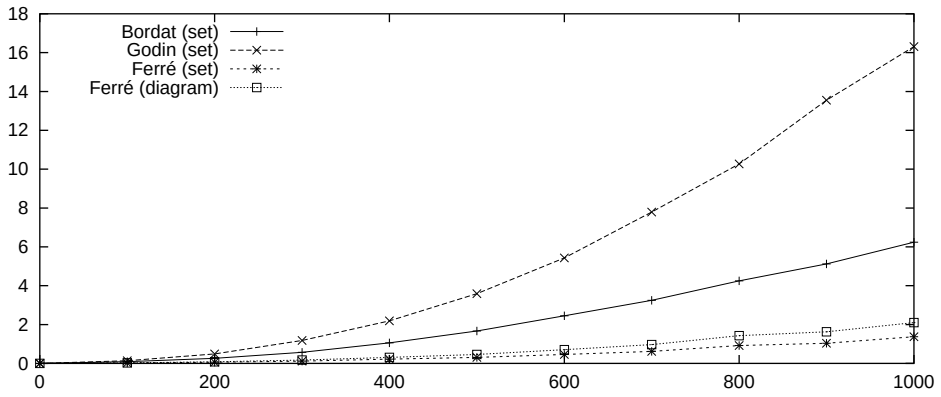

Figure 1: Time (sec) in function of n: $m = 100$, $k = 6$.

Figure 1 shows that the computation time increases with the number of objects. According to Section 4, and if the fact that the number of concepts is in $\mathcal{O}(n)$ when k is fixed, the time complexity is in $\mathcal{O}(n^2)$.

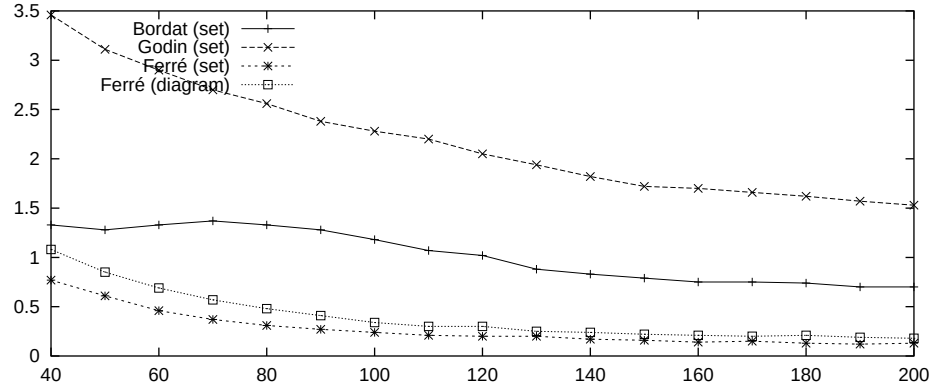

Figure 2: Time (sec) in function of m: $n = 400$, $k = 6$.

Surprisingly, Figure 2 shows that the computation time decreases with the number of attributes. The explanation is that the number of objects n has been used as an over-approximation of the size of extents, while their average size is something like $\frac{kn}{m}$. This effect is the most visible for Bordat's and our algorithms because they are intensively based on extent comparisons.

Figure 3 shows that the computation time is exponential with the number of attributes per object. With Figures 1 and 2 that show that this computation time is not exponential with the other parameters (i.e., n

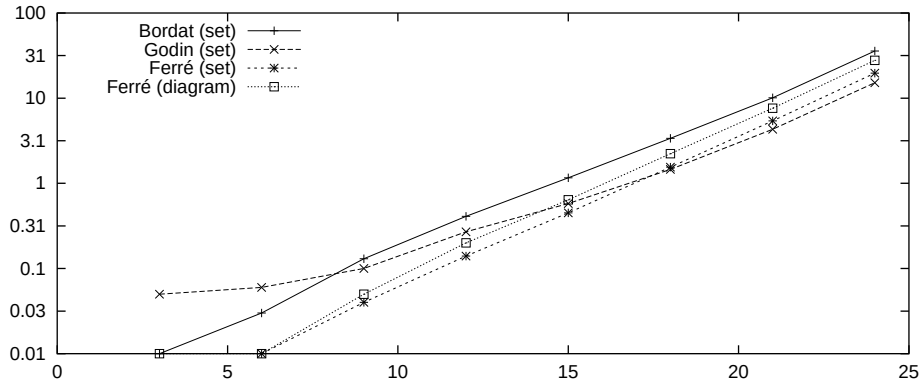

Figure 3: Time (sec) in function of k: $n = 50$, $m = 50$.

and m), this means that the famous exponential nature of concept lattices and their computation only relies in the parameter k. This implies that whenever it is fixed (which is usually the case in real contexts), the time complexity is polynomial in n with a constant factor depending (exponentially) with k.

In conclusion, our algorithm performs better than the others when k is small enough, and this even true when computing the diagram. However, Godin's algorithm, that is the worst for small values of k, becomes the better above some threshold. Of course, this threshold may depend on other parameters n, and m. A question that remains is whether the computation of the diagram by Godin's algorithm would be better or worse than ours in this case?

5.2 Diagonal contexts

Diagonal contexts are square contexts where only the main diagonal is free. This means that $m = n$ and each object has $n - 1$ attributes. This kind of contexts is not realistic, but it is of theoretical interest because their concept lattices are the power set of objects and attributes, and so has an exponential size. It can be considered as the worst case for computation.

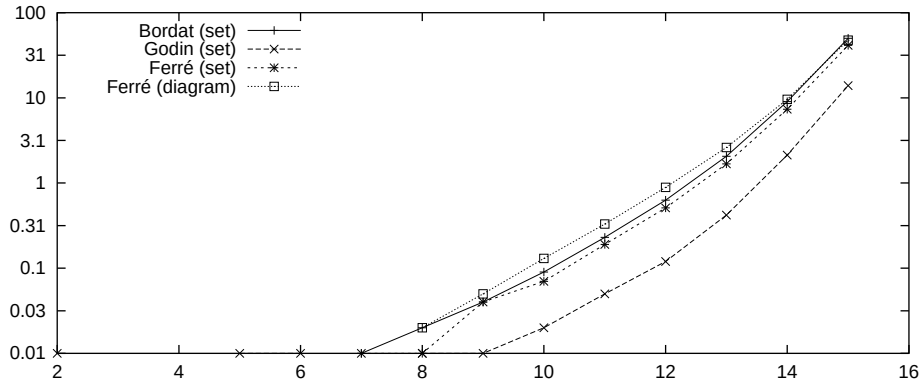

Figure 4: Time (sec) in function of n: $m = n$, $k = n - 1$ (diagonal context).

Figure 4 shows that our algorithm performs worse than Godin's but as well as Bordat's, even when computing the diagram. However, the difference between Godin's and our algorithm is not so big and seems to decrease. Once again, the question of how Godin's compares to us for the diagram computation remains open.

5.3 Logical contexts

In our experiments about logical information systems [FR01], we observed that, while the number of attributes per object is constant in a given application, the number of attributes is growing proportionnaly to the number of objects because each new object brings a few new attributes. This is characteristic of what we call *logical contexts*, where objects are described by a logical formula instead of a set of attributes. There, attributes are *features* automatically extracted from logical descriptions. Hence the production of new attributes. Another specificity of logical contexts is that some attributes are more general than others: a few are shared by all objects while others are specific to one object. We modeled this phenomena by a Zipf law that associates to each attribute a frequency inversely proportional to its rank in the list of all attributes.

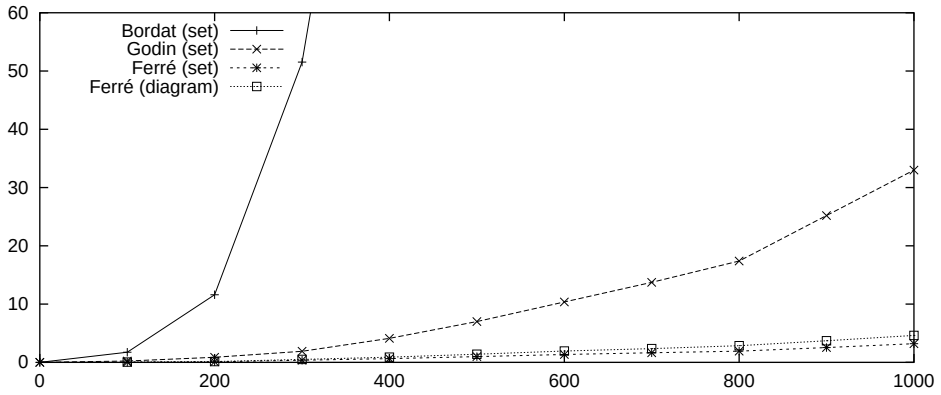

Figure 5: Time (sec) in function of n: $m = 2 \times n$, $k = 6$ (logical context).

Compared to Figure 1 where m is fixed, Figure 5 shows that our algorithm behaves very well in logical contexts, while Godin's and Bordat's are definetely more costly. These results confirm theoretical results of Section 4.

6 Conclusion and Perspectives

We proposed a new algorithm for computing the concept lattice of a context. It combines existing Godin's and Bordat's algorithms. Both theoretical and practical complexity comparisons with these two algorithms prove that our algorithm is the most efficient one in sparse contexts, and the only efficient one in logical contexts. Moreover, it is incremental, which is a necessary condition for some applications.

A future improvement would be to take into account *logical entailments* between attributes, i.e., a taxonomy of attributes, in the generation of sub-concepts (see function `sub-nodes` in Section 3). This would certainly speed up our algorithm, especially in the case of logical contexts.

References

[Bor86] J. Bordat. Calcul pratique du treillis de Galois d'une correspondance. *Mathématiques, Informatiques et Sciences Humaines*, 24(94):31–47, 1986.

[CS00] R. Cole and G. Stumme. CEM - a conceptual email manager. In G. Mineau and B. Ganter, editors, *Int. Conf. Conceptual Structures*, LNCS 1867, pages 438–452. Springer, 2000.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[Fin83] V. K. Finn. On machine-oriented formalization of plausible reasoning in the style of F. Backon–J.S. Mill. *Semiotika Informatika*, 20:35–101, 1983. In Russian.

[FR01] S. Ferré and O. Ridoux. Searching for objects and properties with logical concept analysis. In H. S. Delugach and G. Stumme, editors, *Int. Conf. Conceptual Structures*, LNCS 2120, pages 187–201. Springer, 2001.

[FR02] S. Ferré and O. Ridoux. The use of associative concepts in the incremental building of a logical context. In G. Angelova U. Priss, D. Corbett, editor, *Int. Conf. Conceptual Structures*, LNCS 2393, pages 299–313. Springer, 2002.

[GK00] B. Ganter and S. Kuznetsov. Formalizing hypotheses with concepts. In G. Mineau and B. Ganter, editors, *Int. Conf. Conceptual Structures*, LNCS 1867, pages 342–356. Springer, 2000.

[GM94] R. Godin and R. Missaoui. An incremental concept formation approach for learning from databases. *TCS*, 133(2):387–419, 1994.

[GMA93] R. Godin, R. Missaoui, and A. April. Experimental comparison of navigation in a Galois lattice with conventional information retrieval methods. *International Journal of Man-Machine Studies*, 38(5):747–767, 1993.

[GMA95] R. Godin, R. Missaoui, and H. Alaoui. Incremental concept formation algorithms based on Galois (concept) lattices. *Computational Intelligence*, 11(2):246–267, 1995.

[GW99] B. Ganter and R. Wille. *Formal Concept Analysis — Mathematical Foundations*. Springer, 1999.

[HSWW00] J. Hereth, G. Stumme, R. Wille, and U. Wille. Conceptual knowledge discovery and data analysis. In G. Mineau and B. Ganter, editors, *Int. Conf. Conceptual Structures*, LNCS 1867, pages 421–437. Springer, 2000.

[KO01] S. Kuznetsov and S. Objedkov. Comparing performance of algorithms for generating concept lattice. In E. Mephu Nguifo et al., editor, *ICCS-2001 Int. Workshop on Concept Lattices-based Theory, Methods and Tools for Knowledge Discovery in Databases*, CRIL – IUT de Lens, France, 2001. Stanford University.

[KS94] M. Krone and G. Snelting. On the inference of configuration structures from source code. In *Int. Conf. Software Engineering*, pages 49–58. IEEE Computer Society Press, May 1994.

[Kuz99] S. Kuznetsov. Learning of simple conceptual graphs from positive and negative examples. In J. M. Żytkow and J. Rauch, editors, *Principles of Data Mining and Knowledge Discovery*, LNAI 1704, pages 384–391. Springer, 1999.

[Lin95] C. Lindig. Concept-based component retrieval. In *IJCAI95 Workshop on Formal Approaches to the Reuse of Plans, Proofs, and Programs*, 1995.

[Sne98] G. Snelting. Concept analysis — A new framework for program understanding. *ACM SIGPLAN Notices*, 33(7):1–10, July 1998.

[ST00] G. Snelting and F. Tip. Understanding class hierarchies using concept analysis. *ACM Transactions on Programming Languages and Systems*, 22(3):540–582, 2000.

[VM01] P. Valtchev and R. Missaoui. Building concept (Galois) lattices from parts: Generalizing the incremental methods. In H. S. Delugach and G. Stumme, editors, *Int. Conf. Conceptual Structures*, LNCS 2120, pages 290–303. Springer, 2001.

[Wil82] R. Wille. *Ordered Sets*, chapter Restructuring lattice theory: an approach based on hierarchies of concepts, pages 445–470. Reidel, Dordrecht Boston, 1982.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399